

## APPENDIX G

```

#include <stdio.h>
#include <unistd.h>
#include <errno.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <string.h>
#include "fastdb.h"

int FalseDigs = 0;
int LinksFollowed = 0;

//
// Name:      QueryRecord
//
// Description:
//      Query record routine using double CRC hash index files.
//
//      If SIMPLE_HASH is defined, routine simulates a simple hash
//      search by ignoring the second hash value for test comparision.
//
//      If GET_DIRECT is defined, routine simulates a theoritical
//      optimal solution by translating search key into record offset
//      via compulation and seeks directly to get the record.
//

#ifndef GET_DIRECT

int QueryRecord ( char *key, char *record )
{
    int i;
    unsigned short crc;
    unsigned short crc16;
    unsigned long startoff;
    unsigned long curpos;
    int bytes;
    char *ptr;
    keyindx_t   crcbucket[CRCBUCKETSIZE+1];
    keyindx_t   *keyptr;
    static int keyfd = -1;
    static int datafd = -1;

    // first time file open of keyindx and data file
    if ( keyfd == -1 )
    {
        // open the key index file
        keyfd = open ( "keyindx", O_RDONLY );
        if ( keyfd == -1 )
        {
            printf ( "Can't open keyindx for write. %s\n",
                    strerror ( errno ) );
            exit(1);
        }

        // open the Data file
        datafd = open ( "Data", O_RDONLY );
    }
}

```

```

    if ( datafd == -1)
    {
        printf ( "Can't open Data for read. %s\n",
                strerror ( errno ) );
        close ( keyfd );
        keyfd = -1;
        exit(1);
    }
}

// calculate CRC-CCITT checksum
crcstr ( &crc, (unsigned char *) key );

#ifdef SIMPLE_HASH
    // calculate CRC-16 checksum
    crc16str ( &crc16, (unsigned char *) key );
#endif

// find right bucket
startoff = (unsigned long)crc * (CRCBUCKETSIZE + 1) * sizeof (keyindx_t);

while ( 1 )
{
    // seek to this bucket's offset
    curpos = lseek ( keyfd, startoff, SEEK_SET );
    if ( curpos != startoff )
    {
        printf ( "lseek in keyindx file failed\n" );
        exit(1);
    }

    // read the bucket
    if ( read ( keyfd, (char *)crcbucket, sizeof(crcbucket) )
        != sizeof(crcbucket) )
    {
        printf ( "read from keyindx file failed\n" );
        exit(1);
    }

    // search for crc16 match
    keyptr = crcbucket;
    for ( i = 0; i < CRCBUCKETSIZE ; i++, keyptr++ )
    {
#ifdef SIMPLE_HASH
        // check for crc16 match slot in bucket
        if ( keyptr->crc16 == crc16 && keyptr->offset != MAXOFFSET )
#else
        // simulating simple hash, ignoring crc16 value...
        if ( keyptr->offset != MAXOFFSET )
#endif
        {
            // found match, read record for key verification...
            curpos = lseek ( datafd, keyptr->offset, SEEK_SET );
            if ( curpos != keyptr->offset )
            {
                printf ( "lseek in Data file failed\n" );
                exit(1);
            }
        }
    }
}

```

```

    }

    bytes = read ( datafd, record, MAXRECORD - 1 );
    if ( bytes < 2 )
    {
        printf ( "bad read, bytes = %d\n",
                bytes );
        exit(1);
    }

    record[bytes] = 0;
    ptr = strchr ( record, '\n' );
    if ( !ptr )
    {
        printf ( "error, no new line found\n" );
        exit(1);
    }

    *ptr = 0;

    // get first field, e-mail key is first
    ptr = strchr ( record, '\t' );
    if ( !ptr )
    {
        printf ( "bad read, no tab found\n" );
        exit(1);
    }

    // check record (key is first record)
    *ptr = 0;
    if ( strcmp ( record, key ) == 0 )
    {
        // found record
        *ptr = '\t';
        return 1;
    }
    else
    {
        //printf ( "false dig, found: '%s'\n",
        //record );
        FalseDigs++;
    }
}

// no macth found in this bucket, check if next crcbucket is linked.
if ( keyptr->offset == MAXOFFSET )
    break; // no more buckets, record not found

// follow link to next bucket...
startoff = keyptr->offset;
LinksFollowed++;
//printf ( "Linking to next bucket at 0x%x\n", startoff );
} // while ( 1 )

// record not found
printf ( "record not found. key = %s\n", key );
return ( 0 );

```

```

}

#else

//
// Name:      QueryRecord
//
// Description:
//      Query record routine using theoretical optimal solution.
//      Does not use index file, rather it transform the search
//      key into record offset via computation. Basically,
//      provides baseline performance for random seeks/reads
//      on a large data file.
//
//

int QueryRecord ( char *key, char *record )
{
    int i;
    unsigned long startoff;
    unsigned long curpos;
    unsigned int  rindex;
    int bytes;
    char *ptr;
    static int datafd = -1;

    // first time file open of keyindx and data file
    if ( datafd == -1 )
    {
        // open the Data file
        datafd = open ( "Data", O_RDONLY );
        if ( datafd == -1 )
        {
            printf ( "Can't open Data for read. %s\n",
                    strerror ( errno ) );
            exit(1);
        }
    }

    // translate key into record number
    sscanf ( key, "%x", &rindex );
    startoff = (unsigned long)rindex * 100;

    // seek directly to record and read it
    curpos = lseek ( datafd, startoff, SEEK_SET );
    if ( curpos != startoff )
    {
        printf ( "lseek in Data file failed\n" );
        exit(1);
    }

    bytes = read ( datafd, record, MAXRECORD - 1 );
    if ( bytes < 2 )
    {
        printf ( "bad read, bytes = %d\n",
                bytes );
        exit(1);
    }
}

```

```
record[bytes] = 0;
ptr = strchr ( record, '\n' );
if ( !ptr )
{
    printf ( "error, no new line found\n" );
    exit(1);
}
*ptr = 0;

// get first field, e-mail key is first
ptr = strchr ( record, '\t' );
if ( !ptr )
{
    printf ( "bad read, no tab found\n" );
    exit(1);
}

// check record (key is first record)
*ptr = 0;
if ( strcmp ( record, key ) == 0 )
{
    // found record
    *ptr = '\t';
    return 1;
}
else
{
    printf ( "false dig, found: '%s'\n",
            record );
    FalseDigs++;
}

// record not found
printf ( "record not found. key = %s\n", key );
return ( 0 );
}
#endif // GET_DIRECT solution
```